Using Range-Revocable Pseudonyms to Provide Backward Unlinkability in the Edge

Cláudio Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Portugal
claudio.correia@tecnico.ulisboa.pt

Miguel Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Portugal
miguel.p.correia@tecnico.ulisboa.pt

Luís Rodrigues
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Portugal
ler@tecnico.ulisboa.pt

ABSTRACT

In this paper we propose a novel abstraction that we have named Range-Revocable Pseudonyms (RRPs). RRPs are a new class of pseudonyms whose validity can be revoked for any time-range within its original validity period. The key feature of RRPs is that the information provided to revoke a pseudonym for a given time-range cannot be linked with the information provided when using the pseudonym outside the revoked range. We provide an algorithm to implement RRPs using efficient cryptographic primitives where the space complexity of the pseudonym is constant, regardless of the granularity of the revocation range, and the space complexity of the revocation information only grows logarithmically with the granularity; this makes the use of RRPs far more efficient than the use of many short-lived pseudonyms. We have used RRPs to design EDGAR, an access control system for VANET scenarios that offers backward unlinkability. The experimental evaluation of EDGAR shows that, when using RRPs, the revocation can be performed efficiently (even when using time slots as small as 1 second) and that users can authenticate with low latency ($0.5 - 3.5$ *ms*).

CCS CONCEPTS

• **Security and privacy → Access control**; **Distributed systems security**; *Pseudonymity, anonymity and untraceability*; Trusted computing.

KEYWORDS

privacy, verifier local revocation, backward unlinkability

ACM Reference Format:
Cláudio Correia, Miguel Correia, and Luís Rodrigues. 2023. Using Range-Revocable Pseudonyms to Provide Backward Unlinkability in the Edge. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623111

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11.
https://doi.org/10.1145/3576915.3623111

1 INTRODUCTION

Anonymous authentication offers both accountability and privacy, protecting clients from curious application providers while ensuring that only authorized participants are able to use the application [40, 42, 59, 65]. The number and relevance of applications that require anonymous authentication are increasing. Examples include crowdsensing [33, 56, 64] and Vehicular Networks (VANETs) [6, 32, 60], where clients voluntarily share information about their environment for the common good. Client authentication is a crucial mechanism to provide accountability for malicious and erroneous activity, ensuring the reliability of these applications [33, 58]. Unfortunately, authentication can compromise *user privacy*, as it may be associated with sensitive information, such as location [56]. This is exacerbated by the fact that, in most of these applications, clients are mobile and may need to *authenticate frequently*, e.g., when they move to the range of a different base station or cell. Multiple authentications may be linked to extract additional information such as daily routines [33] or health status [45] for financial gain [3, 51, 69]. Anonymous authentication can be achieved using Group Signatures (GS) schemes [9, 64] or pseudonym certificates [48].

A challenging task in this context is to support *revocation* without violating privacy. Revocation aims to prevent some clients from further authenticating in the system. Client revocation may be required in the event of credential misuse, sensor malfunctioning, change in client privileges, stolen secret keys, or when a client leaves voluntarily. Client revocation can be implemented in different ways. We distinguish two main classes of revocation strategies, namely, *global client revocation* and *verifier local revocation*.

Strategies based on global client revocation require all clients to obtain new credentials (or update their credentials) every time a single client is revoked. Examples of this strategy include Ateniese *et al.* [2] (where the group public key is renewed at each revocation) and Ohara *et al.* [55] (where a small public membership message is broadcast at each revocation). These approaches make revocation very onerous in scenarios with many clients (e.g., consider vehicle numbers in VANETS) and impractical in mobile settings, where clients may become temporarily disconnected from the network.

Strategies based on Verifier Local Revocation (VLR) [9, 11] do not require that all clients are contacted when a given client is revoked. Instead, only the nodes that perform authentication (often called the signature *verifiers*) have to be informed about the revoked clients [40, 62, 64, 65]. In systems that use pseudonyms, this involves sending to the verifiers a Certificate Revocation List (CRL) with the pseudonyms of the revoked client. In systems based on group

signatures, this involves sending a cryptographic token that can be used to trace the digital signatures of the revoked client.

A problem with both approaches is that, if one or more credentials have been used before revocation, an attacker can cross-check the information used for revocation with the information collected when those pseudonyms were used to break the privacy of the client. Ideally, client revocation should not allow linking credentials that have been used prior to the revocation, a property known as *Backward Unlinkability (BU)* [34, 40, 42, 53]. Previous strategies to provide BU assign credentials that are valid only during a given time slot of a certain duration [34, 42, 59, 64]. Then, when a client is revoked, only the credentials for future time slots are revoked and no information is disclosed regarding credentials used prior to revocation. One can divide these recent strategies as GS with time-bounded keys [15, 24] or pseudonyms with time slots [34, 42]. However, these schemes require the use of revocation lists whose size grows linearly with the number of time slots which, in practice, preclude the use of fine-grain time slots.

Revoking only the credentials for future time avoids backward linkability but, unfortunately, if time slots are large, it may be unacceptable to let revoked clients continue accessing resources until the current slot expires. For this reason, many systems immediately revoke the credentials for the current time slot, at the expense of exposing the client's privacy during that period [34, 40, 42, 53]. Our new class of pseudonyms supports efficient revocation even when fine-grain time slots are used, avoiding this dilemma.

This paper contributes to designing anonymous authentication systems for edge computing scenarios, like VANETs, that offer revocation and BU. We make two key contributions in this context:

Contribution 1: We propose a novel abstraction named Range-Revocable Pseudonyms (RRPs). RRPs are pseudonyms that can be revoked for any time-range within their original validity period. Clients hold a number of RRPs that is proportional to the number of authentication actions they need to perform, regardless of the granularity of the linkability window. Each RRP should be used at most once during its lifetime. In runtime, the RRP can be used to generate a capability that is only valid for the specific time slot where the RRP is being used. The key feature of RRPs is that the information provided to revoke a pseudonym for a given time-range cannot be linked with the information provided when using the pseudonym outside the revoked range. In particular, if a pseudonym is revoked at some point in time, it is impossible for an attacker to find out if that pseudonym has been used before that time. We provide an algorithm to implement RRPs where the space complexity of the pseudonym is constant, regardless of the granularity of the revocation range, and the space complexity of the revocation information only grows logarithmically with the granularity; this supports the use of fine-grain slots and makes the use of RRPs far more efficient than the use of many short-lived pseudonyms. We show that RRPs can be used to solve efficiently the BU problem for anonymous authentication.

Contribution 2: We propose an access control system for VANET scenarios [6, 32, 60] that uses RRPs to offer BU. Our access control system, named EDGAR, illustrates how one can leverage RRPs to enforce authentication and revocation. In EDGAR, we deploy Pseudonym Manager (PM) servers that run on the edge of the network, serving clients with new RRPs. Since a PM server holds sensitive information, and the edge infrastructure in VANETS is known to be exposed to attacks [74], we have designed the PM server to be executed with the support of a Trust Execution Environment (TEE), such as Intel SGX enclaves [17]. This allows the server to provide new RRPs to clients without disclosing their identities, even if the untrusted environment is compromised.

EDGAR paves the way for arbitrarily small time slots with minimal overhead. Consider a client who during the day goes to the hospital and before that to a nearby shop. When using EDGAR, clients only need a number of pseudonyms proportional to the number of resources they need to access (in this example, 2 resources), and not proportional to the granularity of the time slots. With previous work, if the two events above could occur within 20 minutes of each other, a client would require 72 pseudonyms; if the events could occur within 5 minutes of each other, previous works could require 288 pseudonyms. Also, with RRPs the cost of revocation is logarithmic with granularity: only 12 credentials would need to be revoked with a 20 minute granularity and only 16 credentials would need to be revoked with a 5 minute granularity.

Our prototype uses SGX-enabled Intel NUC (Next Unit of Computing) nodes that can be configured to serve as verifiers or a PM server. We measured the latency experienced by clients. Our results show that EDGAR can authenticate clients with low latency, in the order of $0.5 - 3.5$ *ms*, which satisfies the requirements of most latency-sensitive applications, such as augmented reality [49] and safety applications [36]. We also compared EDGAR with a related work [34] using a real data set of vehicle traces. We show that EDGAR achieves multiple orders of magnitude in storage savings and that the revocation can be performed efficiently, even when using time slots as small as 1 second.

2 RELATED WORK

Backward Unlinkability has been defined in previous work [34, 40, 42, 53] as follows: *when a revocation occurs, the signatures produced by the client before the revocation interval remain anonymous.* The notion of unlinkability captures the inability of an adversarial server to link a revocation phase of the protocol to any individual signing phase. We are interested in non-blocking approaches such as VLR [9, 11], where the credential of non-revoked clients remain valid, and only the verifiers need to be informed about the credentials of revoked clients. As discussed next, the most popular anonymous authentication schemes that offer BU are based on GS or pseudonyms. In both cases, solutions typically consist of assigning different credentials to different time intervals and then revoking only the credentials for future intervals. Unfortunately, in these previous works, the cost of revocation grows linearly with the granularity of the intervals. Note that if intervals are large, it may be unacceptable to wait for the next interval to revoke the credentials: in this case, it may also be necessary to revoke the credentials for the current interval. Unfortunately, this makes all the credentials used in the current interval vulnerable to being linked.

Anonymous Blacklisting is a term used to describe techniques that are able to safeguard the privacy of revoked clients. Techniques

to ensure this goal include the use of pseudonyms [68], group signatures [63], accumulators [4], and zero-knowledge proofs (ZKPs) [67]. Most systems that aim to offer anonymous blacklisting also aim at offering BU [35], but either use computationally expensive cryptographic operations or also incur a cost that is linear with the granularity of the linkability window. For example, BLAC [67] relies on inherently computationally expensive ZKPs [35]. We avoid the use of ZKPs to implement RRPs due to their high cost; instead, we explore more efficient approaches.

Accumulators, Symmetric Keys, and IBE: Cryptographic accumulators [13] may be vulnerable to linkability [63], i.e., the previously performed operations become linkable when a user is revoked. These solutions also lack VLR since clients need to update their witness at each revocation [4]. Credentials based on symmetric keys [6] require a high level of trust in the verifier and are susceptible to identity theft if the verifier is compromised [37]. Nymble [68] achieves BU but requires a central manager to share a symmetric key with every verifier. Credentials based on Identity Based Encryption (IBE) [8] do not provide anonymity (as they consider the user's identity as the public key) and incur considerable overhead due to expensive cryptographic operations.

Group Signatures with Time-Bound Keys: GS [9] allow different signatures produced by different group members to be verified using a common group public key, achieving anonymity in the set formed by the group members. GS schemes have been augmented with mechanisms to support VLR, as suggested by Brickell [10] and formalized by Boneh and Shacham [9]. Unfortunately, in this scheme, the revocation is performed by publishing a cryptographic token that links all the signatures produced from a revoked member, compromising the anonymity of signatures produced before the revocation. Nakanishi and Funabiki [53] extend [9] to offer BU while preserving VLR. Their approach divides the time into slots and locks a different secret key for each slot, revoking only the keys for current and future slots. Chu *et al.* [15] introduce the notion of Time-Bound Keys (TBK) by setting a configurable expiration date in each key, improving the revocation performance in VLR-GS schemes. In recent years, different solutions have been proposed, following a similar path while aiming to reduce the revocation cost and complexity. LBR [63] requires a trusted online manager to check revocation. Rahaman *et al.* [59] embed pseudoIDs in private key parameters and ties the pseudoID to an epoch, improving the revocation check complexity to $\log(R)$, where R is the size of the revocation list. Emura *et al.* [24] propose an efficient solution with a constant signing cost, but clients are required to download expiration information at each time slot. In Sucasas *et al.* [64], the authors also achieve BU, yet, their solution prevents clients from participating in the same task several times. Ishida *et al.* [40] leverage a mixture of IBE with GS, generating IBE private keys locked to time slots. However, their revocation is still based on the GS private key, also having $O(R\,T)$ (where T is the number of time slots). Despite the interesting properties of GS schemes, GS solutions are usually complex and some may require heavy cryptography operations (such as ZKPs), resulting in few implementations and deployments in real-world scenarios.

Pseudonyms with Bounded Time Slots: Client anonymity can also be achieved by using pseudonyms. These can be implemented using a Public Key Infrastructure (PKI), where clients maintain multiple keys to represent pseudonyms [6, 61, 62]. Pseudonym-based solutions also struggle to offer BU. Some solutions invalidate global information, forcing clients to renew credentials at each revocation [13, 32], failing to preserve VLR. V-token [62], IFAL [71] and PRESERVE [26] follow the C2C-CC standard [26], revoking only the long-term vehicles certificates and letting the pseudonyms expire, also failing the VLR. PUCA [29] requires the owner of the pseudonym to trigger revocation, letting a misbehaving entity evade revocation. The most common solution is to publish all pseudonyms of the revoked client in a CRL [34, 42, 65, 73], respecting VRL but failing BU. The challenge of maintaining the unlinkability of pseudonyms after revocation was first addressed by Haas *et al.* [34], followed by Khodaei *et al.* [42], and implemented in SCMS POC [73] and CAMP [14] pilots, supported by Volkswagen, Mazda, and Nissan. These solutions associate pseudonyms with time intervals and revoke only pseudonyms of the current and future intervals. However, all interactions in the current slot can still be linked and an adversary can use the revocation information to break anonymity [34].

Privacy at the Edge: Edge infrastructures [36], supported by numerous *fog nodes* [70], enable computation near clients. Local authentication within edge resources is vital to meet latency requirements and ensure availability. Privacy is a major concern in VANETs, where vehicles generate and transmit substantial amounts of data. Private companies [69] are exploring ways to monetize user data, often at the expense of privacy, with estimates projecting a worldwide market value of $750 billion by 2030 [69]. For example, unethical edge providers [3, 66] may sell user data to insurance companies, that can subsequently tailor insurance plans based on individual driving habits [54]. Car-sharing and rental agencies can exploit user data with the same purpose [30]. An attacker could also gain access to user data in the edge infrastructure [21], inferring if the certain individual is out of the household or has been attending the hospital [50, 69].

To mitigate these problem, pseudonyms are recommended in the GDPR and by ETSI [27], and are a standard practice in various connected vehicle pilot programs of major car manufacturers (ETSI [25], IEEE [52], NHTSA [54]) such as CAMP [20], New York City [39], and Canada [1] pilots. According to a study by ETSI on the use of pseudonyms [26], frequent pseudonym changes enhance privacy: "*the more often an ITS-S*[1] *changes its pseudonym, the higher its privacy*". However, revoking access rights for a client using different pseudonyms can compromise anonymity when an adversary leverages the revocation information to link the pseudonyms [34]. Approaches that mitigate backward linkability by associating pseudonyms with time slots result in increased storage requirements for pseudonyms at the client side and, consequently, in the CRL [34].

Comparison: Table 1 summarizes the differences between the related work. We highlight that, with our RRPs implementation, the size of the revocation information grows only logarithmically with the number of time slots. Cryptographic accumulators do not offer

[1]Intelligent Transport Systems (ITS) refer to network components, including the On-Board Equipment (OBE) of a vehicle.

Systems	VLR	BU	RDS	Revocation data size for an epoch	PM storage (excluding revocation data)	Client storage for an epoch	Signature size or verification	Revocation communication cost
Cryptographic Accumulators								
PERA [4]	✗	✗	S-b	$R\,T$	N	p	p	R
Camenisch et al. [13]	✗	✗	–	R	N	1	1	R
Symmetric Encryption								
Octopus [37]	✓	✗	–	R	N	1	1	R
Mix Zones [31]	✓	✗	–	R	N	1	1	R
Nymble [68]	✓	✓	S-b	$R\,p\,T$	V	$p\,T$	1	$R\,p$
Identity Based Encryption								
Boneh et al. [8]	✗	✓	–	–	N	1	1	–
Echenim et al. [23]	✓	✗	S-b	R	N	1	1	R
Group Signatures								
Brunger et al. [11]	✓	✗	S-b	$R\,T$	N	1	1	R
Chu et al. [15]	✓	✗	E-b	$R\,\log(T)$	$N\,\log(T)$	$\log(T)$	$\log(T)$	$R\,\log(T)$
Ohata et al. [63]	✗	✓	–	R	R	1	1	–
Emura et al. [55]	✗	✓	S-b	$R\,\log(N/R)$	$N\,\log(N)$	$\log(N)$	1	R
Sucasas et al. [24]	✓	✓	S-b	$R\,T$	N	1	1	R
Nakanishi et al. [59]	✓	✓	S-b	$R\,p\,T$	$(p+1)\,N$	1	1	$R\,p$
Ishida et al. [33]	✓	✓	S-b	$R\,p\,T$	N	1	1	$R\,p$
Rivezzi et al. [61]	✓	✓	S-b	$R\,T$	N	1	1	R
PRESERVE [26]	✓	✓	S-b	$R\,T$	N	1	1	R
Public-key Encryption								
IPA [71]	✓	✗	S-b	$R\,p\,T$	$N\,p$	$p\,T$	1	$R\,p$
Vtoken [62]	✗	✓	–	–	N	$p\,T$	1	–
PASS [65]	✗	✓	–	–	N	$p\,T$	1	–
SCMSPOC [73]	✗	✓	S-b	$R\,p\,T$	1	$p\,T$	1	–
Khodaei et al. [34]	✓	✓	S-b	$R\,p\,T$	N	$p\,T$	1	R
Haas et al. [42]	✓	✓	S-b	$R\,p\,T$	N	$p\,T$	1	R
RRPs/EDGAR (Ours)	✓	✓	S-b	$R\,p\,T$	N	$p\,T$	1	R
	✓	✓	E-b	$R\,p\,\log(T)$	1	p	$\log(T)$	$R\,p\,\log(T)$

N: Number of clients. R: Number of revoked clients. p: Required pseudonyms.
RDS: Revocation Data Structure. S-b: Slot-based. E-b: Epoch-based.
T: The number of time slots in one epoch. PM: Pseudonym Manager. V: Number of verifiers.

Table 1: Properties and complexity offered by different systems, we omit $O()$ notation for simplicity.

VLR. Symmetric encryption cannot protect user privacy from the verifier, and IBE schemes revoke clients by timeout [8] (failing VLR) or issuing the user identifier in a CRL [23] (breaking anonymity). When using GS, clients can generate multiple unlikable signatures with the same secret key, achieving $O(1)$ for the client storage. Some of these GS schemes revoke clients without providing BU, by simply publishing a token for all possible signatures [11, 15]. Other solutions cannot provide the VLR property [55, 63]. Although GS schemes that offer both BU and VLR simultaneously require the revocation procedure to manage a number of credentials that is proportional to the number of time slots $O(R\ p\ T)$ [59, 64] and $O(R\ T)$ [24, 40, 53]. Another limitation of GS schemes is that they rely on complex and heavy cryptographic operations, in particular, to support revocation; this can induce large latencies when performing authentication. PKI based schemes are appealing due to their cryptographic efficiency and wide adoption. Some PKI schemes delay the revocation until all pseudonyms expire [26, 62, 71], breaking VLR. Previous schemes that provide both VLR and BU suffer from the same issue as GS scheme [34, 42, 65, 73], by locking pseudonyms to time slots, they require revocation information that is linear with the time slots, times each pseudonym, $O(R\ p\ T)$. In addition, these solutions require the clients to carry pseudonyms for all time slots, imposing a storage burden of $O(p\ T)$.

3 SYSTEM MODEL

This section presents preliminaries on RRPs and EDGAR, which is an edge authentication system based on RRPs. In EDGAR, to perform authentication, a client first uses an RRP to obtain a *capability*. This capability, which is only valid for a given target time slot, is

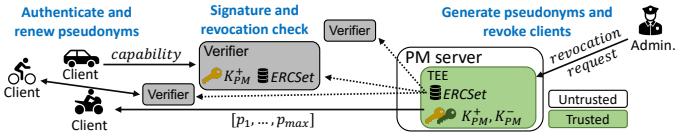

Figure 1: System Overview.

then presented to a verifier. To revoke the use of a RRP during a range of time slots, the corresponding capabilities are revoked. There is a level of indirection between the RRPs assigned to clients and the capabilities used for authentication and revocation that is the enabler to achieve backward unlinkability.

3.1 Entities

EDGAR is composed of three main types of entities: clients, verifiers, and a (distributed) pseudonym manager service. We follow a nomenclature similar to previous work [34, 59].

Clients: the application client that generates signatures to perform authentication against any verifier. Clients are the holders of RRPs that they use to generate capabilities to ensure anonymity. Clients are responsible for renewing their RRPs when needed.

Verifiers: the component that performs client authentication before granting access to a resource. Verifiers are responsible for checking the capabilities provided by clients before granting access. They are also responsible for updating their state by fetching the list of revoked capabilities from the manager servers.

Pseudonym Manager (PM): this component is responsible for providing new RRPs to clients and, when necessary, revoking capabilities generated from these pseudonyms. PM servers are the only entity capable of accessing the true identity of a client so, in our implementation, they run partially inside a TEE, ensuring users' anonymity even if the device is compromised.

Administrator: a trusted entity responsible for adding clients to the system and request PMs to revoke clients.

Figure 1 shows the interactions between these entities, where the PM uses a TEE. The figure represents a typical collective perception scenario, where mobile devices are used to extend human perception. In this example, mobile devices (the clients) authenticate towards the verifiers, to update or download information. When required, clients can contact a nearby PM to renew the set of RRPs used to generate capabilities. Periodically, verifiers will pull from the PM updated revocation information.

3.2 Fault Model

We assume a partial synchrony model [22]. In this model there are unstable periods when messages may be arbitrarily delayed, and stable periods when messages between correct entities arrive within at most Δ units of time. Additionally, we assume that correct processes have access to loosely synchronized clocks, which can differ at most by ϵ. We assume that at most f server nodes can be faulty. We do not place constraints on the number of faulty clients.

Furthermore, verifiers and clients are insecure and prone to Byzantine faults [43]. PM servers are executed (partially) inside TEEs and are only subject to crash and omission faults. Thus, faulty

Notation	Definition	Notation	Definition
t/epoch	Large time interval	s	Time slot, part of an epoch
δ	Duration of a time slot s	c	Capability
K^-, K^+	Private and public key	p	Pseudonym
e^s	Node label for the time slot s	ERCSet/erc	Encoded revoked capability sets
l_x	Latchkeys for e^s	i	Identifier of pseudonym from a user in an epoch
h	Latchkey tree height	M	Extra pseudonyms to circumvent false positives
f	Number of faulty nodes	cid	Client identifier
m	Bloom filter size (bits)	n	Number of items inserted in a Bloom filter
k	Number of hash or index functions	c_s	Number of clients in EDGAR
f_r	Fraction of pseudonyms to be revoked	d	Branching factor of the latchkey tree
x	False positive rate	I	Maximum number of pseudonyms
N	Number of PM replicas		a client can possess

Table 2: Table of notations.

clients may use expired or invalid credentials when contacting servers, faulty verifiers may arbitrarily deny or grant access to resources, but faulty PMs will never provide faulty information, and will never renew pseudonyms for revoked clients. EDGAR ensures liveness during stable periods and offers graceful degradation during unstable periods: when the network is unstable and nodes are unable to receive up-to-date information in a timely manner, they may stop providing service, but never compromise safety.

3.3 Threat Model

We trust only administrators and PMs. Following related work [34, 62], an administrator is responsible for adding and revoking users in the system by contacting the PM server. We assume that each PM has a processor with TEE (e.g., Intel SGX), as shown in Figure 1. All other entities within the system are considered untrusted and susceptible to the control of attackers, potentially engaging in malicious activities. Table 2 provides the notations.

Malicious Client: may attempt to generate pseudonyms or capabilities to impersonate a valid client and access resources to which it is not authorized. It can also try to use old capabilities and pseudonyms after being revoked to authenticate towards verifiers.

Malicious Verifier: if a verifier is compromised, the resource that the verifier is protecting becomes unprotected, but this is not the problem we consider in this paper. For example, under a DoS attack, a verifier may be unable to refresh revocation information and should enter a "safe-mode" (the safe-mode behaviour is application specific but may be as simple as halting). The problem we consider is that a malicious verifier may try to perform linking attacks [62], by associating (linking) different pseudonyms with a single client, breaking user anonymity. This attack becomes trivial when revocation lists that contain all pseudonyms of a client are published [34]. A malicious verifier may collect all the information/data that it observes, e.g., with the objective of deducing user identity.

Malicious Pseudonym Manager: PM code is split in two parts, one that runs inside the TEE and one that runs outside the TEE. The latter can be compromised and engage in malicious behavior, supporting many of the previously introduced attacks. The untrusted part of PM may attempt to modify, delay, block, or read all messages on the system. This behavior may be done in collusion with other entities to facilitate Linking Attacks or allow a user to evade revocation. Furthermore, we assume that a node suffering from denial of service (DoS) is one of the f faulty nodes and that at most f servers can be faulty.

Trust Assumptions: Entities use asymmetric key pairs to establish secure channels. Clients employ RRPs for authentication, integrity, and non-repudiation. Both the PM and the administrator hold unique key pairs, (K^-_{PM}, K^+_{PM}) and $(K^-_{admin}, K^+_{admin})$, respectively, being both public keys known to all entities. Specifically, the administrator's public key K^+_{admin} is hard-coded in the enclave's source code. We assume that the PM correctly executes our protocol within the TEE, where K^-_{PM} remains securely within the enclave. The PM will only revoke users if instructed by the trusted and authenticated administrator, and will generate fresh pseudonyms for non-revoked and authenticated clients. We assume that there is no collusion between the trusted PM and the verifiers.

The communication between the administrator and the enclave is based on a PKI using their keys. We assume a trusted administrator who only revokes pseudonyms after informing the corresponding clients. Although supporting revocation auditability is beyond the scope of this paper, we discuss different approaches to extend EDGAR and ensure revocation auditability in Section 5.10. Furthermore, both capabilities and revocation information are accompanied by a digital signature created using K^-_{PM}, confirming the origin from the PM TEE.

In our work, we make the usual assumptions about the security of TEEs/enclaves [17] (code/data executed/stored inside the TEE have integrity and confidentiality guaranteed), about the cryptographic schemes (they satisfy their security properties) and cryptographic keys (secret and private keys are never disclosed). In the prototype, we use Ed25519 to generate digital signatures [7]. As a collision-resistant hash function, we use SHA-256. We use Intel SGX as our TEE, although our scheme can be easily adapted to other TEEs. We use the Intel SGX SDK inside the enclave and OpenSSL outside.

Although side-channel attacks such as Foreshadow and LVI [12] exist, we consider the defense from these attacks to be orthogonal to our contribution; possible mitigations are discussed in Bagher*et al.* [5]. Correctly synchronizing concurrent data structures can mitigate exploits against synchronization bugs [72], with the help of debugging checkers[2] [46].

4 RANGE-REVOCABLE PSEUDONYMS

RRPs are a novel abstraction that provides authentication based on pseudonyms whose validity can be revoked for any time-range within their original validity period. Clients hold a number of RRPs that is proportional to the number of authentication actions they need to perform. A validity of an RRP is bounded to an *epoch*. An epoch is divided into time *slots* of length δ. The parameter δ is application-specific but can be small, e.g., 1 minute or less. An epoch is assumed to be much larger than the slot, e.g., 1 day. Each RRP should be used for authentication at most once. To perform authentication, a client instantiates a *capability* that is specific to target slot. If a client is revoked for a time period, pseudonyms are not revoked directly; instead, only the capabilities associated with the time-slots of that period are revoked. We store these capabilities in an *Encoded Revoked Capability Sets* (ERCSet). An RRP can be revoked for a short period, by revoking only the capabilities associated with an interval of time-slots, or permanently, by

[2]In EDGAR implementation only a Bloom filter and the current epoch value are accessed concurrently inside the enclave.

revoking the capabilities associated with all future time-slots. Since the revocation information is connected, indirectly, by capabilities, to pseudonyms, when using RRP, a client is required to carry a different RRP for each access it needs to perform. However, any given RRP can be used at any slot of the epoch. Thus, the number of RRPs a client needs to keep is independent of the granularity of the time-slots. This contrasts with previous pseudonym-based solutions, where clients need to carry a number of pseudonyms that grows linearly with the *epoch granularity* (i.e., the number of slots in an epoch).

4.1 Overview

Authentication based on RRPs uses 3 different related objects, namely (range-revocable) *pseudonyms*, (time-bound) *capabilities*, and ERCSet. At an abstract level, the operations supported by these objects are the following:

- $p^{epoch} \leftarrow$createRRP(*cid, epoch,* K^-_{PM}): used to create a new RRPs, that can be used by client *cid* during a target *epoch*. Only PMs, using their private key K^-_{PM}, can create RRPs.
- $c^s \leftarrow$getCapability(p^{epoch}, s, K^-_p): used to create a capability associated with an RRP p^{epoch} for time slot s (s must belong to the epoch for which the pseudonym was created). Only PMs and the client that owns the pseudonym, and the correspondent private key K^-_p, can create capabilities.
- *boolean* $\leftarrow$verifyCapability(c^s, K^+_{PM}): To verify if a capability was generated from a valid RRPs, used by verifiers during authentication, requires the PM public key K^+_{PM}.
- *ERCSet* $\leftarrow$createERCSet(*capabilities*): used only by PMs to create an ERCSet that encodes one or more given capabilities, using some one-way function, such that it is unfeasible to extract a capability from the ERCSet. These capabilities are filtered to ensure that they do not compromise unlinkability (Section 4.3).
- *ERCSet* $\leftarrow$mergeERCSet(erc_1, erc_2): used to merge two ERCSets so that a single ERCSet can be used to capture the revocation of multiple capabilities. PMs and verifiers can merge ERCSets.
- *boolean* $\leftarrow$isRevoked(*erc*, *capability*): used to verify if a capability is part of an ERCSet. This operation is used by verifiers to check if a capability has been revoked.

The manager creates RRPs on request from authorized clients. If later an RRP needs to be revoked for a given range of time slots, the PM generates the corresponding capabilities and encodes them in an ERCSet that is sent to the verifiers.

Clients hold a small number of RRPs (e.g., corresponding to the number of distinct events), and instantiated a short-lived capability (for the current slot) to authenticate. Then, it presents the capability to the verifier. The verifier checks if the capability is correctly constructed, is *genuine* (i.e., if it was generated from a valid RRPs) and subsequently check if the capability has not been revoked; only in this case, the client is granted access to the resource.

To ensure unlinkability, a client must never present two capabilities generated from the same RRP, as capabilities generated from the same RRP can be linked (cf. Section 4.3). Therefore, clients have to carry a number of RRPs proportional to the number of resources they need to access. However, contrary to previous systems, the revocation of an RRP for a time-slot does not expose capabilities that may have been used in non-revoked time slots: this is guaranteed by the use of a one-way function to encode revoked capabilities.

4.2 Making Range-Revocation Efficient

A problem with the use of time-bound pseudonyms is that the number of pseudonyms that need to be revoked grows with the granularity of the time slots. RRPs are not immune to this problem, because to revoke the use of an RRP in a range of time slots, all capabilities associated with those time slots need to be encoded in the ERCSet. However, our implementation of RRPs uses a mechanism that allows the revocation cost to grow only logarithmically with the granularity, rather than linearly, as previous approaches.

To achieve this goal, a capability is represented by a sequence of *latchkeys*, extracted from a set of latchkeys that are associated with a given RPP. The latchkeys are organized in a tree of fanout d, such that there is a leaf latchkey for each individual time slot on an epoch (in this paper, we use $d = 2$, i.e., binary latchkey trees). Figure 2 provides a simple example where a binary tree of latchkeys is associated with an epoch of 1 hour divided in 4 time-slots of 15 minutes. Note that the latchkey tree structure resembles but is not a Merkle tree: the tree nodes are generated independently (the value of a parent node does not depend on the value of its children).

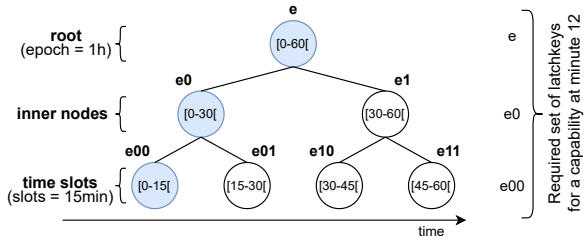

Figure 2: Latchkeys for time slot $[0, 15[$ of epoch $[0, 60[$.

A capability for a given time slot is represented by the set of latchkeys in the path from the root of the tree to the corresponding leaf node in the tree. Using the example of Figure 2, the capability for the first slot would be represented by the following set of latchkeys: $\{e, e0, e00\}$. Note that each capability, for each time slot, is always different, because it contains one unique leaf latchkey. However, different capabilities may have some latchkeys in common; in particular, all capabilities include the root latchkey.

A capability is only considered *valid* if all latchkeys used to represent it are valid. Therefore, the capabilities can be revoked by invalidating any of its latchkeys. In particular, a capability for a given time slot can be revoked by invalidating the leaf latchkey associated with that slot. However, it is also possible to revoke multiple latchkeys by invalidating latchkeys that are inner nodes of the tree: by invalidating an inner node, all the capabilities that are part of the sub-tree rooted at that inner node are invalidated. This can also be illustrated using our example. Consider that the client is revoked at the beginning of the second slot ($e01$) until the end of the epoch. At this point, the client may already have used its pseudonym p to generate a capability to access the resource during the first slot. To prevent linkability, the latchkeys used in the first

slot cannot be revoked, i.e, the latchkeys e, $e0$ and $e00$ cannot be revealed. To revoke all future capabilities that may be generated with pseudonym p, it suffices to revoke latchkeys $e01$ and $e1$. Note that the capabilities generated for the second slot must use $e01$, and the capabilities generated for the third/fourth slots must use $e1$.

We use this construction to perform revocation efficiently. ERCSets do not explicitly contain capabilities, but only latchkeys that belong to those capabilities and a single latchkey can be used to revoke multiple capabilities. It is easy to show that the number of latchkeys that need to be revoked is at most $\log_d$ with the number of slots. In fact, for each pseudonym valid in the epoch, the number of latchkeys will be given by $\log_d(granularity)$.

4.3 RRP Implementation

We now describe the construction of RRPs.

Scheme assumptions: We assume that the epoch and time slot size are publicly known to all parties in the system. This means that any party can independently and consistently calculate all the labels from any leaf to the root (i.e., e, e^0, etc.), as illustrated in Figure 2. There is also a maximum number of pseudonyms I that any client can use in any given epoch.

Cryptographic primitives: We assume there are sources of entropy and a function that allow generating random asymmetric key pairs (K^-, K^+). We assume that there is a function, named DetKeyGen(*seed*), to generate asymmetric key pairs deterministically from a *seed* value. There is also a deterministic signature scheme that, given some private key K^- and a *text* as input, will output a deterministic signature sig = DetSign(K^-, *text*). The output *sig* can be verified by *true*/*false* = VerSign(K^+, *text*, *sig*). Lastly, there is a secure one-way function Digest(*text*) that computes a digest on the *text* input and is not possible to invert given the output.

PM Keys: There is an asymmetric key pair (K^-_{PM}, K^+_{PM}) associated with every PM. The private key K^-_{PM} is only known by the PMs and is kept in the implementation inside the TEE enclave. The public key K^+_{PM} is known to all participants, including clients and verifiers.

RRPs: An *RRP* is a tuple $\langle cid, epoch, i, K^-_p, K^+_p, sig_p\rangle$ where *cid* is the client identifier (only known by the client and the PM), *epoch* is the time windows for which the pseudonym is valid, i is a label that can be used to distinguish each pseudonym instance generated for the same epoch, where $i \in [1, I]$. The (K^-_p, K^+_p) is a unique asymmetric key pair associated with the pseudonym, and sig_p is the signature performed with the private key of the PM over the concatenation of the epoch, and public key of the pseudonym, sig_p = DetSign(K^-_{PM}, $epoch \,\|\, K^+_p$). Note that some fields of an RRP are secrets known only to the client and PM and never revealed to a verifier. In particular, only the client and the PM know the secret key K^-_p associated with a given pseudonym. To obtain an RRP, a client establishes a secure channel with a PM, presents its client identifier *cid*, and obtains one or more RRPs for some given target *epoch*. When describing EDGAR, we will discuss for which epochs clients are allowed to obtain RRPs from a PM.

Generating (K^-_p, K^+_p)***:*** The asymmetric key pair associated with a pseudonym is generated using the DetKeyGen(*seed*) primitive. We use as seed the tuple $\langle cid, epoch, i\rangle$, avoiding the need for the PM to memorize the information associated with all the pseudonyms it created, as it can always re-create them (as explained below, the key pair is also needed to perform revocation). Recall that *cid* is known only by the client and the PM. This identifier is securely stored by the PM inside the enclave. Also, DetKeyGen is non-reversible, thus two different public keys created for different epochs and/or instances for the same client cannot be linked with the secret *cid*.

Latchkeys: Latchkeys are unique for each pseudonym and are obtained by deterministically signing the label of the corresponding node with the private key K^-_p of the pseudonym. Therefore, the latchkey l_0 associated with the label node e^0 of an *RRP*, is generated as l_0 = DetSign(K^-_p, e^0), and can be verified by using the public key of the pseudonym by performing VerSign(K^+_p, e^0, l_0).

Capabilities: A capability c for a given time slot s is a tuple:

$$c = \langle K^+_p, sig_p, l_{leaf}, \ \ldots, \ l_{00}, \ l_0, \ l_{root}\rangle$$

where K^+_p is the public key of the pseudonym and the latchkeys correspond to the nodes on the path from the root of the latchkey tree to the leaf latchkey node associated with the time-slot s. Note that a capability has a number of latchkeys that is logarithmic with the granularity of the time-slots in the epoch. The latchkeys that are part of a capability can be generated on demand, when the capability is created, and are not required to be stored explicitly by the client. It should also be noted that any two capabilities generated from the same RRP reveal the same K^+_p and can be linked; therefore, a client that wants to prevent authorization request to be linked should always use different RRPs.

To verify a capability, a verifier performs the following steps. First, it uses the public key of the PM to verify sig_p, calculating VerSign(K^+_{PM}, $epoch \,\|\, K^+_p$, sig_p). Then, it uses K^+_p to verify if the latchkeys presented with the capability are in fact associated with that RRP, by performing VerSign(K^+_p, e^x, l_x). If *all* latchkeys can be verified using K^+_p and follow a correct path from the current slot to the root, the capability is genuine. Note that a capability can be genuine but may have been revoked, as explained next.

ERCSet: an ERCset in an encoding of a set of latchkeys that represents a set of revoked capabilities. The set of latchkeys encoded in an ERCset has the following properties: *inclusion-of-revoked* – if a capability has been revoked, at least one of its latchkeys is encoded in the ERCSet; *exclusion-of-non-revoked* – if a capability has not been revoked, none of its latchkeys are encoded in the ERSet. Below we explain how latchkeys are selected to be encoded in the ERSet to satisfy these properties. Latchkeys are encoded in the ERCSet using a one-way function, Digest(l_x) . Thus, verifiers can check if a given latchkey belongs to an ERCSet but cannot extract latchkeys from the ERCSet. Different data structures that rely on one-way functions could be used to implement ERCSet, including SHA256, or compact data structures such as Cuckoo filters [28], Cascade filters [44] or Count-min sketch [16]. We use Bloom filters to implement the ERCSet. Bloom filters are efficient and, as discussed later, a good fit for the EDGAR architecture. A disadvantage of Bloom filters is that they can present false positives, but we will explain later how EDGAR circumvents this limitation.

Revoking a single capability: To revoke a capability c_p of a pseudonym p, the PM encodes in the ERCSet the leaf latchkey l_x associated with c_p. This trivially satisfies the *inclusion-of-revoked* and *exclusion-of-non-revoked* properties: the encoded latchkey belongs to the revoked capability but does not belong to any other capability (each capability is associated with a distinct, unique, leaf latchkey).

Revoking a range of capabilities: Revoking a set of capabilities of a pseudonym could be trivially achieved by encoding the corresponding set of leaf latchkeys, but this would have a linear cost. The latchkey hierarchy is used to reduce this cost as follows. Let d be the fanout of the latchkey tree. Any d latchkeys that have the same parent in the latchkey tree can be replaced by their parent. This also satisfies the *inclusion-of-revoked* and *exclusion-of-non-revoked* properties of ERCSets: 1) the parent of any latchkey is part of the capability that includes that latchkey and 2) a parent latchkey is not included in capabilities other than the capabilities that include its children. The susbtitution of all d sibling latchkeys by their parent latchkey can be applied recursively in the tree. Note that the root latchkey can only be included in an ERCSet when a pseudonym is revoked for the entire duration of the epoch, because the root latchkey belongs to all capabilities for that epoch. Appendix A provides a precise description of this algorithm with pseudo-code.

Merging ERCSet: An advantage of using Bloom filters is that ERCSet can be easily merged by performing bitwise OR operations. This makes it easy to disseminate revocation lists for many different RRPs in a single data structure.

Checking if capability revoked: Verifiers receive ERCSets from PMs, store them, and use them to check if the capabilities presented by clients have not been revoked. After checking if a capability is genuine, verifiers test if any of the latchkeys are included in the most recent ERCSet. If even a single latchkey is in the ERCSet, the capability is considered revoked.

4.4 RRPs Linkability Analysis

A key problem with previous approaches for performing pseudonym revocation is that the information used for revocation could be linked with the information used for access control (in particular, this is obvious when the pseudonym identifier is used both for authentication and revocation). This allows an adversary to collect information about the resources that have been accessed by revoked pseudonyms. When a client has several pseudonyms that are revoked together, an attacker can link the past usage of these pseudonyms to break the privacy of the user. RRPs avoid this problem because the information used for revocation cannot be linked with the information used for access control. Thus, if a client used one or more pseudonyms prior to revocation, the use of these pseudonyms cannot be linked based on the revocation data.

Here we present an argument that RRPs offer unlinkability.

Observation 1. *Verifiers cannot generate latchkeys associated with a pseudonym.*

Basis: Latchkeys are generated using the private key K_p^- of the pseudonym. The private key is generated using the secret *cid* that is shared between the PM and the client and never revealed to other entities. Therefore, verifiers cannot generate latchkeys. □

Observation 2. *ERCSets do not include latchkeys used outside the revocation interval.*

Basis: This property is achieved by construction, that ensures the exclusion-of-non-revoked. As described above, when a PM assembles an ERCSet, it never includes in the ERCSet latchkeys that are part of capabilities for time-slots outside the revocation interval. □

Argument 1. *Revocation information cannot be linked with authorization information used outside of the revocation interval.*

Basis: The revoked latchkeys are encoded in an ERCSet using $\textsc{Digest}(l_x)$, so verifiers cannot extract latchkeys from an ERCSet. Verifiers can only test if a given latchkey has been revoked. However, by Obs. 1, verifiers cannot generate latchkeys, so they can only test latchkeys that are provided by the client when presenting a capability. By Obs. 2, latchkeys for capabilities associated with non-revoked time-slots are not included in an ERCSet. □

Argument 2. *Capabilities generated from different pseudonyms cannot be linked.*

Basis: All the information in a capability depends on the asymmetric key pair associated with the pseudonym. Asymmetric key pairs for different pseudonyms are different because they are generated using different seeds (the unique instance number i is part of the seed $\langle cid, epoch, i \rangle$). Additionally, asymmetric key pairs cannot be linked to the seed used for generation (this derives directly from the standard properties of DetKeyGen). □

5 EDGAR

We now present the design of an anonymous authentication system for the edge that leverages RRPs to offer backward unlinkability. We have named our system EDGAR: EDGe distributed Access contRol, targeting the VANET scenario [6, 32, 60]. The goal of EDGAR is to reduce the linkability window, improving client privacy at the edge. EDGAR demonstrates how to use our RRPs abstraction and how to address implementation challenges in a distributed setting.

5.1 EDGAR in VANETs

In the VANET scenario, vehicles continuously broadcast CAM messages [26] containing various information such as their geolocation, sensor readings, direction, and speed. This information is crucial for various edge applications, including enhanced navigation, traffic congestion estimation, remote vehicle diagnostics, autonomous cars, and others [30]. However, as explained in Section 2, edge providers can collect and monetize this data, at the expense of users privacy, highlighting the importance for clients to use anonymous authentication methods such as EDGAR. We now contextualize the RRPs entities to the corresponding entities in EDGAR:

Clients: These are vehicles that constantly propagate CAM messages with location and sensor readings, with the purpose of enhancing their safety and that of others.

Verifiers: Mainly compose by Roadside Units (RSUs) [19] that listen to all CAM messages, aggregate them, and broadcast them in the network. These devices can be deployed by various local entities (e.g., municipal authorities) or edge providers to improve traffic flow, pedestrian safety, and provide services to vehicles such as infotainment or software updates.

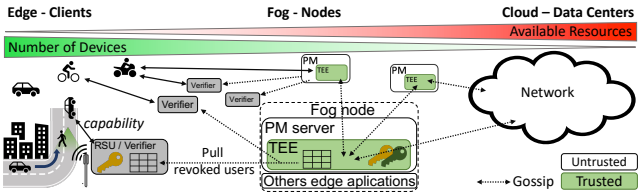

Figure 3: EDGAR entities and VANETs interaction at the edge.

PM servers: These are fog nodes with the same services as verifiers but with higher computational capacity and storage, and may be physically more distant than verifiers. Any fog node with TEEs can serve as a PM. We assume the same trust level as mentioned in Section 3.3, where the PM does not share its private key or the client pseudonyms. However, the PM is controlled by edge providers, who can access the non-trusted zone outside the TEE.

Administrator: In the context of the edge, the administrator should be a trusted entity independent of all applications and providers within the edge. It should work similarly to the current Certificate Authorities (CAs) in PKI.

Figure 3 illustrates the interactions in the edge environment. In this example, a vehicle (a client) presents a capability to the RSU (the verifier). If the capability is valid, the RSU accepts the message from the client and alerts the vehicles about pedestrians behind the corner. Authentication is critical to avoid false information that may cause other drivers to break without justification. The RSU relies on multiple nearby fog nodes, running edge replicas of the PM, to update its state. Vehicles can contact a nearby PM replica to renew the pseudonyms used to generate capabilities, if necessary.

EDGAR prevents the de-anonymization of clients based on the tracking the RRP usage (i.e., even if edge providers like Verizon, Akamai, and Amazon aggregate data from verifiers and the untrusted zone of the PM). Our pseudonyms also protect users' privacy in case of data leaks from verifiers.

5.2 Revocation in EDGAR

Although RRPs supports the revocation of a pseudonym in any range of time slots, in EDGAR we assume that clients can be revoked at a target *revocation time slot* (RTS), selected by the administrator, and that all capabilities of that client are revoked for all time slots after RTS (i.e., the revocation range spans from RTS to the end of the epoch). Also, after being revoked, clients become unable to obtain new pseudonyms from edge PMs after some time. In particular, as we will show later, EDGAR is able to provide the following guarantee: if a client is revoked in a given epoch t, that client may still attempt to use pseudonyms it has obtained for epoch $t + 1$ but will not be able to obtain pseudonyms for epoch $t + 2$.

5.3 Epochs, RRPs, and ERCSet

Time is divided in epochs and epochs are divided in time slots. The length of an epoch and the granularity δ of the time slot are application specific. As we show in the evaluation, the efficient revocation mechanism of RRPs, based on the latchkey hierarchy, supports the use of relatively large epochs and fine-grain granularity, for instance, epochs of one day and time slots of 1 minute.

There is a limit I of the number of pseudonyms that a client can request for a given epoch. When using RRPs this is not a limitation because clients only need to have a pseudonym for each access regardless of the time slot where the pseudonym is used (and not a different pseudonym for each time slot, as in previous work). Also, clients are only allowed to obtain pseudonyms for the current epoch and for the next epoch (we allow clients to obtain in advance pseudonyms for the next epoch to avoid having PMs to be overload with a rush of requests whenever an epoch begins). This allows us to limit the number of pseudonyms that need to be added to ERCSet when a client is revoked. Also, verifiers only accept requests that use pseudonyms from the current epoch. This allows to garbage collect revocation information from previous epochs safely.

Due to the constraints described above, EDGAR is only required to maintain two ERCSets: one associated with the current epoch and another associated with the next epoch. When an epoch t terminates, the ERCSet associated with epoch t can be discarded and a fresh ERCSet is created for the next future epoch ($t + 2$).

5.4 ERCSet dissemination

The revocation of a client is initiated in the central PM. The PM first generates all possible pseudonyms that the client may have obtained for the current epoch (i.e., by creating the pseudonyms for all instances $1 \ldots I$) and creates an ERCSet that revokes all the capabilities that may be generated for these pseudonyms in the range starting from the *revocation time slot* (RTS) to the end of the epoch. For this, it uses the algorithm described in Section 4.2. It then merges this ERCSet into the global $ERCSet_t$ for the current epoch. The PM then generates all possible pseudonyms that the client may have obtained for the next epoch and creates an ERCSet that revokes these pseudonyms for the entire epoch (this is very efficient, because it suffices to include the root latchkey of each pseudonym in ERCSet); it then merges this ERCSet in the global $ERCSet_{t+1}$.

Disseminating client revocation among the PMs. The updated values of $ERCSet_t$ and $ERCSet_{t+1}$ are then disseminated in the system using a two-step procedure. First, they are disseminated from the central PM to all edge replicas of the PM. Then, verifiers pull these values from their nearest PM replicas. EDGAR implements the propagation of ERCSets among PM replicas using a gossip-based broadcast protocol. The central PM first selects $f + 1$ edge PMs at random and sends them the updated ERCSets. When receiving an ERCSet from another replica, a PM checks if the ERCSet is different from the local version. If the ERCSet is the same, it discards the redundant update. If the ERCSet is different, it assumes that it may contain new information and merges it with its own ERCSet, picks other $f + 1$ edge PMs at random, and sends them the updated ERCSets. This eager push strategy allows revocation information to be propagated quickly on the network. Additionally, a PM that does not receive any updates for more than a predefined gossip timeout engages in pull-gossip with another random PM. Pull gossip is used to recover from temporary crashes or disconnections. A PM that is down when a revocation is eagerly propagated will later obtain the information using pull gossip. Note that the ERCSet for a given epoch always accumulates new information. Thus, any

single gossip exchange with an up-to-date server will convey all the information that a node may have missed while disconnected.

Disseminating latchkey revocations to the verifiers. The edge will consist of many verifiers placed at different locations. It is not efficient to have all these PM servers sending the same information to all verifiers. Therefore, we only use pull-gossip to propagate ERCSets to each verifier. Each controller periodically picks a PM at random, pulls $ERCSet_t$ from that server, and merges its content with a local copy of $ERCSet_t$. If a verifier fails to execute the pull-gossip procedure (possibly due to an adversary jamming the PM), it enters "safe-mode" (the specific behavior varies depending on the application, but could involve stopping the service to protect the resource).

5.5 Obtaining New Pseudonyms

Clients can obtain pseudonyms from edge PMs. EDGAR does not require edge PMs to keep an explicit list of all clients that have been revoked and of their corresponding *revocation time slot* as this is already encoded in the ERCSet. When requesting new pseudonyms, a client establishes a secure channel with any edge PM and provides its own *cid* and a valid capability. If the client has not been revoked, the PM can provide the requested pseudonyms for the current of for the next epoch. If the client has been revoked, it will be denied access to additional pseudonyms.

To check if a client has not been revoked, a PM performs the following checks: first it verifies if the capability presented by the client is in fact associated to a pseudonym of that client. This procedure leverages that fact that any PM can create all pseudonyms of a client, and therefore, can check if the public key included in the capability corresponds to a public key of one of the valid pseudonyms for that client. Then, it checks if the capability has not been revoked. If the request passes these tests, the PM generates and sends the requested pseudonyms to the client.

5.6 Size of ERCSets

EDGAR uses Bloom filters to implement ERCSets. Bloom filters have $O(1)$ insertion and query time [47], are space-efficient, and can be merged easily. However, Bloom filters suffer from false positives and should be used with care. In fact, there is evidence that, if not used properly, the false positives generated by Bloom filters can jeopardize the operation of large-scale systems[44]. We first discuss how the size of the Bloom filters used to implement ERCSets is chosen in EDGAR. Later, we discuss how we deal with the fact that false positives cannot be entirely avoided.

The false positive rate of a Bloom filter depends on the filter size m (bits), the number of items to be inserted n, and the number k of hash or index functions used for insertion and search. The false positive rate can be approximated as described in [34]:

$$P(\textit{false positive}) = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \quad (1)$$

In the case of ERCSets, the average number of items in the Bloom filter is given by:

$$n = c_s \times I \times f_r \times \log_d\left(\frac{\textit{epoch}}{\delta}\right) \quad (2)$$

where c_s is the number of clients, I is the average number of pseudonyms that each client uses, f_r is the fraction of pseudonyms that may need to be revoked, d is the branching factor of the latchkey tree, *epoch* is the length of an epoch, and δ is the length of the time slot. By using Eq. 2 to compute the number of latchkeys that are expected inserted in an epoch in a Bloom filter, we can use Eq. 1 to select the size of the Bloom filter that limits the probability of having a false positive to some pre-defined threshold.

Let us assume a scenario with $c_s = 250,000,000$ clients (the estimated number of vehicles in the USA), and assume a fraction of pseudonyms that need to be revoked of 10^{-4} per year (from [34]). If we set the length of an epoch $24h$, this provides an average of revocations per epoch of $f_r = 10^{-4}/365$. Then, if we set the granularity of the time slots to $\delta = 10$ minutes. This yields 144 time slots per epoch. If we use a binary tree of latch keys, the average number of latchkeys used per revocation is $\log_2(144)$. If we assume that clients need at most 10 pseudonyms per day, the expected resulting number of items to be added to the Bloom filter is:

$$n = 250,000,000 \times 10 \times (10^{-4}/365) \times \log_2(144) \approx 4911$$

In this scenario, a Bloom filter of $9KB$ provides a false positive rate of 0.1% (from Eq. 1). We can determine the false positive rate of a capability by: $P_{FP}(C) = 1 - (1 - x)^h$, where x is the false positive rate of the Bloom filter, from Eq. 1, and h is the tree height. This corresponds to a false positive of 6 in every 1000 capabilities. Additionally, if one wants to increase the granularity of the time slot to $\delta = 1$ minute, the number of time slots per epoch increases 10 times, but the number of latchkeys increases only logarithmically, thus the size of the ERCSet must increase only by a factor of $\log_2(1,440)/\log_2(144) = 1.46$, i.e, an ERCset of $13KB$ will be enough to maintain the same false positive rate

5.7 Circumventing False Positives

Even if the size of Bloom filters is set appropriately, there is always some probability of the occurrence of false positives. In EDGAR we bypass this problem by having clients request M extra pseudonyms, in addition to those that are strictly needed to access the resources. If a false positive occurs, the system automatically picks another unused pseudonym and resubmits the authorization request to the verifier. The only perceived effect by the client is an additional latency in serving the request. We show below that the number of additional pseudonyms that a client needs to carry to circumvent the occurrence of false positives is small. Equation 3 describes the probability that a client will execute all authentication successfully with the help of the M extra pseudonyms.

$$P(\textit{full access}) = 1 - \sum_{j=M+1}^{p+M} {}^{p+M}C_{M+1} \times (P_{FP}(C))^j \times (1 - P_{FP}(C))^{p+M-j} \quad (3)$$

When applying Equation 3 to the previous scenario, where $\delta = 10$ minutes, and setting $M = 0$, it is possible to derive that 1% of the clients may fail some authentication; this number can be reduced to $1.9 \cdot 10^{-12}$ by setting $M = 4$. These extra 4 pseudonyms will require increasing the filter size from $9KB$ to just $13KB$ (to achieve the same probability with $M = 0$ would require increasing the size of the Bloom filter by $34KB$). If each client would require 1000 pseudonyms instead, the probability of a client successfully

executing all authentication with $M = 0$ is just 63% but when using $M = 15$ it increases to $1 - (1.9 \cdot 10^{-14})$, with a storage increase from $883KB$ to $896KB$ (to achieve the same probability with $M = 0$ would require a filter of $3.89MB$).

Leveraging M extra pseudonymous is a space-efficient solution to make the effect of false positives negligible, even for large-scale systems such as EDGAR. We could consider alternative encoding techniques that completely eliminate false positives, such as cascade filters [44]. However, this would require anticipating all possible false positives in order to create the multiple filter levels; in a scenario of millions of vehicles with multiple pseudonyms, this operation would be very expensive and might become infeasible.

5.8 Handling Epoch Changes and Quarantine

When a client is revoked, all future capabilities that can be generated from the pseudonyms it may have obtained are revoked. As discussed above, if a client is revoked in epoch t, this requires revoking capabilities for future time slots in epoch t and all the capabilities for epoch $t + 1$. Capabilities for epoch $t + 2$ and other future epochs do not need to be revoked because EDGAR ensures that a client that is revoked in epoch t cannot obtain pseudonyms for epoch $t+2$. This property is guaranteed by a coordination phase that is executed by any PM when it transitions from epoch t to epoch $t+1$. The purpose of the coordination phase is to ensure that any PM that enters in epoch $t + 1$ is aware of all revocations performed in epoch t and, therefore, will refuse to issue pseudonyms for epoch $t + 2$ to clients that have been revoked in epoch t. During coordination, a PM enters in a quarantine mode, where it cannot serve pseudonym requests for epoch $t + 2$.

The coordination protocol is implemented by forcing each PM to send to every other PM its version of ERCsets_t and ERCsets_{t+1} at the beginning of the quarantine. Furthermore, a PM waits to receive revocation information from at least $N - f$ PMs before ending the quarantine. Because revocation is performed by updating $f + 1$ PMs, and a PM waits for the input of other $N - f$ PMs, for each revoked client, a PM is guaranteed to receive at least one up-to-date ERCset that includes the corresponding revoked capabilities. At the end of the quarantine, a PM is guaranteed to be fully aware of all revocations that have occurred in epoch t and can start serving requests for pseudonyms in epoch $t + 2$.

5.9 Handling a PM failure

The temporary failure or disconnection of a PM is treated as follows. When the PM server recovers, it will immediately start the pull-gossip procedure. Eventually, it will be able to get up-to-date information on the revoked clients. The same applies to temporarily disconnected PMs. A PM that is offline for a short period of time can operate normally, even if it is slightly outdated. If it is contacted by a verifier, it will not be able to provide the most recent revocation information, but the verifier will be able to fetch that information from another PM in the next gossip interaction. If it is contacted by a revoked client, it may issue new pseudonyms to that client for the current or the next epoch. However, the corresponding latchkeys for those pseudonyms have already been revoked by other PMs, and the client will be revoked in a bounded time.

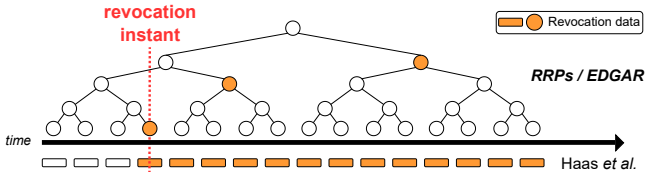

Figure 4: Amount of revocation data: EDGAR vs [34].

5.10 Revocation Auditability

Informally, revocation auditability refers to a user's ability to verify its revocation status at a service provider before attempting to authenticate. As mentioned in Section 3.3, for the current prototype we assume that the administrator informs the clients before proceeding with the revocation. As future work, we plan to augment the system with additional strategies to support revocation auditability. One approach is to use contract-based revocation [35], where the contract semantics are agreed upon by both the user and the provider. This enables the user to determine whether a certain action will constitute misbehavior before deciding whether to engage in it. Another approach has been implemented in Nymble [68]. To ensure that fresh revocation information reaches the client, revocation lists must be published at regular Δi time intervals, containing a signature with the corresponding timestamp. When a client communicates with a verifier, it can first request the list, which must have a fresh signature for the current Δi, and then check if it has not been revoked, otherwise it should halt the authentication process.

5.11 Discussion

In this section, we discuss the key features of EDGAR.

Epoch Based Pseudonyms: Pseudonyms in EDGAR are bound to epochs instead of slots. Capabilities are bound to slots, but can be generated at any moment by the client. This decoupling allows clients to store only the desired number of pseudonyms based on application logic, instead of the δ granularity of slots.

Space Efficiency: Both edge computing and TEEs have memory constraints, making space efficiency a crucial aspect [17]. With EDGAR, revoking a client only requires a logarithmic number of latchkeys, while previous solutions require a linear amount of revocation information relative to the number of time slots (see Figure 4).

Backward Unlinkability: EDGAR revokes future capabilities while ensuring that these capabilities cannot be linked to capabilities used in the past. Unlike previous work, the δ granularity of the time slots can be arbitrarily reduced without imposing a burden on the system: the number of pseudonyms used by a client does not depend on δ and the cost of revocation is $\log_2(1/\delta)$.

Support for Distributed Fault-tolerance: EDGAR distributes and replicates the PM functionality. This increases both availability and resilience. It increases availability because clients can obtain pseudonyms from any correct PM. It increases resiliency, because the coordination required to change epoch effectively prevents PMs that have been isolated or whose clock has been attacked from providing new pseudonyms to revoked clients in future epochs (if the clock is moved backward in time and the server generates

invalid pseudonyms for old epochs; if the clock is moved forward in time, the server cannot progress through quarantine).

Traceability and Accountability: If required, EDGAR can be extended with mechanisms in which verifiers share (limited) information with the PM to provide traceability and accountability. Specifically, a verifier may present one or more used capabilities to the administrator and ask to revoke or trace the anonymous client that is responsible for such capabilities. Depending on the application and the facts to justify the request, the administrator may agree and forward the capabilities to the trusted central PM. Since the PM can generate all the public keys associated with the pseudonyms it has provided, it can subsequently match the used capabilities with the clients identifiers (note that only the trusted PM can perform this operation; this does not conflict with ensuring unlikability, which aims at preventing non-trusted entities, such as verifiers, from achieving the same goal). However, we have not implemented or evaluated such extensions as part of this work.

6 EVALUATION

We evaluate the power of RRPs, using a prototype of EDGAR. We compare the space efficiency of EDGAR against a state-of-the-art scheme for BU in the PKI setting. We also show that our scheme offers a latency suitable for edge applications. Finally, we evaluate EDGAR's throughput when serving pseudonyms. The source code is available at https://github.com/claudio-correia/RRP-EDGAR.

We have implemented both a verifier and a PM server on an Intel NUC10i7FNB. An Intel NUC is an example of what a fog node might be, as it possesses modest computational resources but is relatively inexpensive for large-scale deployments. It has an Intel i7-10710U CPU with Intel SGX, 16GB RAM, and Ubuntu 20.04 LTS. We run the Intel SGX SDK Linux 2.13 Release, Intel SSL-SGX [38] version Linux 2.14_1.1.1k and OpenSSL 1.1.1k. We used a real-world data set composed of multiple vehicle trajectories in the city of Porto [41].

6.1 Space Efficiency

We have experimentally compared EDGAR with Haas *et al.*, as both support backward unlinkability by dividing the epoch in time intervals in the PKI setting. Haas *et al.* scheme was more recently implemented in the SCMS POC pilot [14] under the name of linkage values technique. The comparison is not trivial since the pseudonyms in Haas *et al.* are locked to a time slot, being invalid if used in any other, while in RRP the pseudonyms are free to be used at any moment of an epoch.

For a clear comparison, we test both mechanisms in a real-world use case of a taxi company operating in the city of Porto, using a dataset of taxi trajectories [41]. We choose the mix zones strategy [6] for pseudonym changes, i.e., taxis change pseudonyms at crossroads. This use case requires a large number of pseudonyms due to constant vehicle movement, favoring the Haas *et al.* design. In scenarios with fewer pseudonyms needed over the same period, EDGAR will outperform Haas *et al.* by even larger margins.

Figure 5 shows the results obtained in our experiment. The top part of the figure presents the user latency experienced when acquiring all pseudonyms for a specific epoch, and the bottom part presents the required Bloom filter size for each solution.

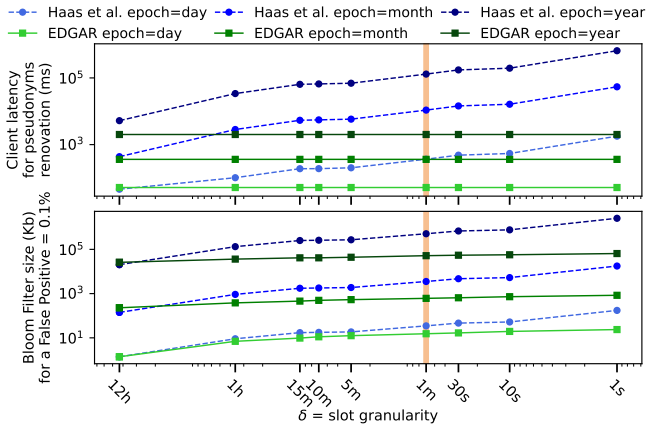

Figure 5: Storage and latency in Haas *et al.* Vs EDGAR.

In the dataset, we observed that some taxis drive long distances, requiring 692 pseudonyms per day (which entails 5,677 per moth and 32,142 per year). Since EDGAR clients can use pseudonyms freely, these values were used as I, the number of pseudonyms instances in an epoch. On the other hand, Haas *et al.* must fix the number of pseudonyms in each δ interval, e.g., with $\delta = 1$ min there was a taxi driver who crossed 12 intersections, forcing to fix 12 different pseudonyms for each δ interval, which has an explosive effect on the number of pseudonyms generated. Furthermore, as we tighten the δ interval, the difference between Haas *et al.* and EDGAR is more pronounced, for $\delta = 1$ sec and epoch = 1 year we observe an improvement of $\approx 2.5 * 10^6$KB to $\approx 6.4 * 10^4$KB, two orders of magnitude lower in the required storage. This results from the logarithmic effect provided by RRPs, while Haas *et al.* suffers from a linear effect. For large values of δ, when an epoch is divided into a few time slots, Haas *et al.* slightly outperforms EDGAR, since the overhead imposed by the latchkey hierarchy is no longer compensated by a significant reduction in pseudonyms.

Finally, we also observed that a large number of taxi trips take around 10 min., so we believe that $\delta = 1$ min would be a reasonable configuration for this use case, representing a storage saving between 35KB and 15KB (for epoch = 1 day) and 508MB to 51MB (for epoch = 1 year) by implementing EDGAR instead of Haas *et al.*, highlighted with the vertical orange line.

6.2 δ Granularity vs Latency Trade-off

The use of the latchkey hierarchy makes revocation of a range of time slots efficient, because a single latchkey can be used to revoke many capabilities. The efficiency of revocation comes at the cost of penalty in the authentication procedure, because the verifier must check a number of latchkeys equal to the tree height (instead of verifying only the leaf latchkey). Fortunately, the height of the latchkey tree grows only logarithmically, and latchkeys can be verified in parallel. Therefore, the penalty of RRPs on latency is small. Figure 6 shows the latency of the verification procedure as the granularity of the time slots increases. For instance, a system that uses $\delta = 1$ min and an epoch of one day requires a binary latchkey tree of depth 11; if the epoch is increased to a month, the depth of the binary tree increases to 16. In such a case, clients

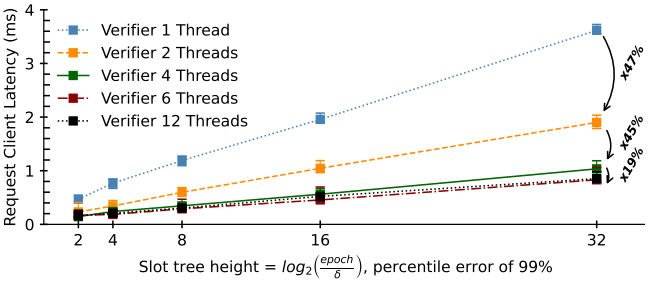

Figure 6: Client latency for capability verification, for different levels of granularity and number of threads.

will incur a latency below 2*ms* in a single-thread verifier, still an acceptable latency for edge applications. Note that a tree with 32 levels can support an extreme large number of time slots, such as the ones resulting from using $\delta = 1$ sec and epoch = 70 years; even in this case, clients would experience only 3.5 ms of latency. We have used multithreading to parallelize the verification of latchkeys, reducing the impact on latency. We observe a latency reduction that is linear with the number of cores of 47% and 45% from 1 to 2 threads and from 2 to 4 threads, respectively. Hyperthreading, from 6 to 12 threads, offers no improvement since most of the time is spent in cryptographic operations, and each core has a single ALU.

6.3 PM Server Throughput

We use the Ed25519 scheme [7] to obtain deterministic digital signatures, but is not yet available in the SGX SDK. Since the PM is responsible for generating the pseudonyms and runs inside the enclave, we implemented two different versions of the Ed25519 inside the enclave: a portable one [57] and one based on OpenSSL [38].

The portable version is straightforward to implement in any type of TEE, but the lack of optimization affects its performance. In the second implementation, we use the Intel SSL-SGX library to import the OpenSSL library into the enclave. This library was designed for the SGX enclaves, being the most efficient implementation of the scheme. We also evaluated the system with and without SGX, using the OpenSSL library outside the enclave as well.

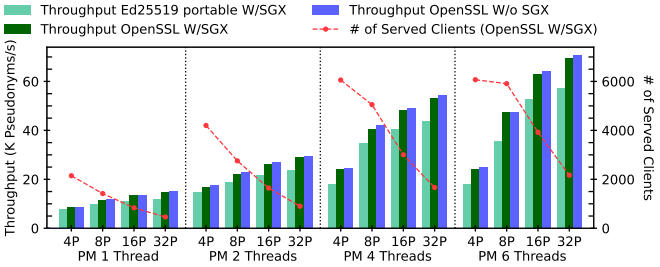

Figure 7: Pseudonym throughput and number of served clients by the PM w/ and w/o SGX. P is the number of pseudonyms generated in each request.

Figure 7 presents the pseudonym throughput for each of these implementations. We vary the number of threads on the PM node side, improving throughput as expected. Additionally, we vary the number of pseudonyms that each client requests from the PM, from 4, 8, 16, and 32 new pseudonyms. We observe that increasing the requested pseudonyms also increases throughput by reducing other system overheads, such as signing and encrypting. On the contrary, the number of served clients decreases since the PM server requires more time for each request. Following the previous discussion and fixing $\delta = 1$ min, a taxi could request 32 new pseudonyms in just 133 ms, useful for at least the next 32 min.

7 CONCLUSION

We presented Range-Revocable Pseudonyms, an abstraction that supports an anonymous authentication scheme based on pseudonyms that is able to enforce backward unlinkability with storage costs that are multiple orders of magnitude lower than those of the related work. The gains derive from our novel technique to decouple pseudonyms from time slots, and authentication/revocation procedures based on the use of latchkeys that can be generated from a given pseudonym for any desired time slot. This technique prevents clients from having to store a large number of unnecessary pseudonyms. As a proof of concept, we have designed and implemented a prototype of EDGAR, an authentication system for the edge based on the use of RRPs. We have used this prototype to perform an experimental evaluation using a real dataset of vehicle traces. The results show that EDGAR is capable of offering low latency and storage savings when serving clients. We motivated our work using a VANET scenario, as one of the most prominent use cases for anonymous authentication at the edge computing environment. Still, many other applications, such as crowdsensing, supply chain tracking, augmented reality, etc., also require anonymity and may benefit from our scheme.

We have made available an extended version of this paper [18] that includes a more detailed proof of correctness and a complete workflow of all the operations discussed in the paper.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and also Fred B. Schneider and Carlos Ribeiro for their valuable comments that greatly helped to improve this paper. This work was supported by the Fundação para a Ciência e a Tecnologia (FCT) scholarship 2020.05270.BD, by national funds through FCT via the INESC-ID grant UIDB/50021/2020 and via the SmartRetail project (ref. C6632206063-00466847) financed by IAPMEI, and by the European Union ACES project, 101093126.

REFERENCES

[1] American Association of State Highway and Transportation Officials. 2014. *National Connected Vehicle Field Infrastructure Footprint Analysis, Final Report.* Retrieved 2022-11-28 from https://ntlrepository.blob.core.windows.net/lib/52000/52600/52602/Cnct_Veh_Footprint_20181017.pdf

[2] Giuseppe Ateniese, Dawn Song, and Gene Tsudik. 2002. Quasi-Efficient Revocation of Group Signatures. In *Proceedings of the International Conference on Financial Cryptography.* Southampton, Bermuda.

[3] AT&T. 2022. *Cybersecurity Insights Report: Securing the Edge.* Retrieved 2022-11-28 from https://cdn-cybersecurity.att.com/docs/industry-reports/cybersecurity-insights-report-eleventh-edition.pdf

[4] Man Au, Patrick Tsang, and Apu Kapadia. 2008. PEREA: Practical TTP-Free Revocation of Repeatedly Misbehaving Anonymous Users. *ACM Transactions on Information and System Security* (2008), 1–34.

[5] Kassem Bagher and Shangqi Lai. 2023. SGX-Stream: A Secure Stream Analytics Framework In SGX-enabled Edge Cloud. *Journal of Information Security and Applications* 72 (2023).

[6] Alastair Beresford and Frank Stajano. 2004. Mix Zones: User Privacy in Location-Aware Services. In *Proceedings of the IEEE Conference on Pervasive Computing and Communications Workshops*. Orlando, FL, USA.

[7] Daniel Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of cryptographic engineering* (2012), 77–89.

[8] Dan Boneh and Matthew. Franklin. 2001. Identity-Based Encryption from the Weil Pairing. In *Proceedings of the Annual International Cryptology Conference*. London, UK.

[9] Dan Boneh and Hovav Shacham. 2004. Group Signatures with Verifier-Local Revocation. In *Proceedings of the ACM conference on Computer and communications security*. Washington, DC, USA.

[10] Ernie Brickell. 2003. An efficient protocol for anonymously providing assurance of the container of a private key. *Submitted to the Trusted Computing Group* (2003).

[11] Julien Bringer and Alain Patey. 2011. Backward Unlinkability for a VLR Group Signature Scheme with Efficient Revocation Check. *Cryptology ePrint Archive* (2011).

[12] Jo Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lippi, Marina Minkin, Daniel Genkin, Yuval Yarom, Berk Sunar, Daniel Gruss, and Frank Piessens. 2020. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *Proceedings of the IEEE Symposium on Security and Privacy*. San Francisco, California.

[13] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *Proceedings of the International Cryptology Conference*. Santa Barbara, California, USA.

[14] Camp, LLC. 2016. *Security Credential Management System Proof-of-Concept Implementation—EE Requirements and Specifications Supporting SCMS Software Release 1.1.* Technical Report. Vehicle Safety Communications Consortium.

[15] Cheng-Kang Chu, Joseph Liu, Xinyi Huang, and Jianying Zhou. 2012. Verifier-Local Revocation Group Signatures with Time-Bound Keys. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*. Seoul, Korea.

[16] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* (2005), 58–75.

[17] Cláudio Correia, Miguel Correia, and Luís Rodrigues. 2020. Omega: A Secure Event Ordering Service for the Edge. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*. València, Spain.

[18] Cláudio Correia, Miguel Correia, and Luís Rodrigues. 2023. Using Range-Revocable Pseudonyms to Provide Backward Unlinkability in the Edge (Extended Version). *arXiv preprint arXiv:2308.03402* (2023).

[19] Marco Correia, João Almeida, Paulo Bartolomeu, José Fonseca, and Joaquim Ferreira. 2022. Performance Assessment of Collective Perception Service Supported by the Roadside Infrastructure. *Electronics* (2022), 347.

[20] Crash Avoidance Metrics Partners LLC (CAMP). 2021. *Technical Publications*. Retrieved 2022-11-28 from https://www.camplc.org/

[21] Dark Reading. 2021. *Cybercriminals Take Aim at Connected Car Infrastructure*. Retrieved 2022-11-28 from https://www.darkreading.com/attacks-breaches/cybercriminals-take-aim-at-connected-car-infrastructure

[22] C. Dwork, N. Lynch, and L. Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* (1988), 288–323.

[23] Sebastián Echeverría, Dan Klinedinst, Keegan Williams, and Grace Lewis. 2016. Establishing Trusted Identities in Disconnected Edge Environments. In *Proceedings of the IEEE/ACM Symposium on Edge Computing*. Washington, DC, USA.

[24] Keita Emura, Takuya Hayashi, and Ai Ishida. 2017. Group Signatures with Time-Bound Keys Revisited: A New Model, an Efficient Construction, and its Implementation. *IEEE Transactions on Dependable and Secure Computing* (2017), 292–305.

[25] European Telecommunications Standards Institute. 2015. *ETSI TS 103 097 V1.2.1:Intelligent Transport Systems (ITS); Security; Security header and certificate formats.* Retrieved 2022-11-28 from https://www.etsi.org/deliver/etsi_ts/103000_103099/103097/01.02.01_60/ts_103097v010201p.pdf

[26] European Telecommunications Standards Institute. 2018. *ETSI TR 103 415 V1.1.1: Intelligent Transport Systems (ITS); Security; Pre-standardization study on pseudonym change management.* Retrieved 2022-11-28 from https://www.etsi.org/deliver/etsi_tr/103400_103499/103415/01.01.01_60/tr_103415v010101p.pdf

[27] European Telecommunications Standards Institute. 2021. *ETSI TS 102 941 V1.4.1: Intelligent Transport Systems (ITS); Security; Trust and Privacy Management.* Retrieved 2022-11-28 from https://www.etsi.org/deliver/etsi_ts/102900_102999/102941/01.04.01_60/ts_102941v010401p.pdf

[28] Bin Fan, Dave Andersen, Michael Kaminsky, and Michael Mitzenmacher. 2014. Cuckoo Filter: Practically Better Than Bloom. In *Proceedings of the ACM International on Conference on emerging Networking Experiments and Technologies*. Sydney, Australia.

[29] David Förster, Frank Kargl, and Hans Löhr. 2014. PUCA: A Pseudonym Scheme with User-Controlled Anonymity for Vehicular Ad-Hoc Networks (VANET). In *Proceedings of the IEEE Vehicular Networking Conference (VNC)*. Paderborn, Germany.

[30] Freedom of Information and Privacy Association (FIPA). 2019. *The Connected Car: who is in the driver's seat?* Retrieved 2022-11-28 from https://fipa.bc.ca/wp-content/uploads/2018/01/CC_report_lite.pdf

[31] Julien Freudiger, Maxim Raya, Márk Félegyházi, Panos Papadimitratos, and Jean-Pierre Hubaux. 2007. Mix-Zones for Location Privacy in Vehicular Networks. In *Proceedings of the ACM Workshop on Wireless Networking for Intelligent Transportation Systems*. Vancouver, Canada.

[32] Carlos Gañán, Jose Munoz, Oscar Esparza, Jorge Mata-Díaz, and Juanjo Alins. 2015. EPA: An efficient and privacy-aware revocation mechanism for vehicular ad hoc networks. *Pervasive and Mobile Computing* (2015), 75–91.

[33] Raghu Ganti, Fan Ye, and Hui Lei. 2011. Mobile Crowdsensing: Current State and Future Challenges. *IEEE communications Magazine* (2011), 32–39.

[34] Jason Haas, Yih-Chun Hu, and Kenneth Laberteaux. 2011. Efficient Certificate Revocation List Organization and Distribution. *IEEE Journal on Selected Areas in Communications* (2011), 595–604.

[35] Ryan Henry and Ian Goldberg. 2011. Formalizing Anonymous Blacklisting Systems. In *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland, California, USA, 81–95.

[36] Yun Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. 2015. Mobile Edge Computing — A key technology towards 5G. *ETSI white paper* 11, 11 (2015).

[37] Maged Ibrahim. 2016. Octopus: An Edge-Fog Mutual Authentication Scheme. *International Journal of Network Security* (2016), 1089–1101.

[38] Intel Corporation. 2017. *Intel Software Guard Extensions SSL.* Retrieved 2022-11-28 from https://github.com/intel/intel-sgx-ssl

[39] Intelligent Transportation Systems Joint Program Office. 2021. *Connected Vehicle Pilot Deployment Program.* Retrieved 2022-11-28 from https://www.its.dot.gov/pilots/overview.htm

[40] Ai Ishida, Yusuke Sakai, Keita Emura, Goichiro Hanaoka, and Keisuke Tanaka. 2018. Fully Anonymous Group Signature with Verifier-Local Revocation. In *Proceedings of the International Conference on Security and Cryptography for Networks*. Amalfi, Italy.

[41] Kaggle. 2015. *Data Set ECML/PKDD 15: Taxi Trajectory Prediction.* Retrieved 2022-11-28 from https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/

[42] Mohammad Khodaei and Panos Papadimitratos. 2018. Efficient, Scalable, and Resilient Vehicle-Centric Certificate Revocation List Distribution in VANETs. In *Proceedings of the ACM conference on security & privacy in wireless and mobile networks*. Stockholm, Sweden.

[43] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* (1982), 203–226.

[44] James Larisch, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, and Christo Wilson. 2017. CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers. In *Proceedings of the IEEE Symposium on Security and Privacy*. San Jose, CA, USA.

[45] Mu Lin, Nicholas Lane, Mashfiqui Mohammod, Xiaochao Yang, Hong Lu, Giuseppe Cardone, Shahid Ali, Afsaneh Doryab, Ethan Berke, Andrew Campbell, et al. 2012. BeWell+ Multi-dimensional Wellbeing Monitoring with Community-guided User Feedback and Energy Optimization. In *Proceedings of the conference on Wireless Health*. 1–8.

[46] Bozhen Liu and Jeff Huang. 2018. D4: Fast Concurrency Debugging with Parallel Differential Analysis. *ACM SIGPLAN Notices* (2018), 359–373.

[47] Lailong Luo, Deke Guo, Richard Ma, Ori Rottenstreich, and Xueshan Luo. 2018. Optimizing Bloom Filter: Challenges, Solutions, and Comparisons. *IEEE Communications Surveys & Tutorials* (2018), 1912–1949.

[48] Anna Lysyanskaya, Ronald Rivest, Amit Sahai, and Stefan Wolf. 1999. Pseudonym Systems. In *Proceedings of the International Workshop on Selected Areas in Cryptography*. Ontario, Canada.

[49] Simone Mangiante, Guenter Klas, Amit Navon, Zhuang GuanHua, Ju Ran, and Marco Silva. 2017. VR is on the Edge: How to Deliver 360° Videos in Mobile Networks. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. Los Angeles, CA, USA.

[50] Ed Markey. 2015. Tracking & Hacking: Security & Privacy Gaps Put American Drivers at Risk. *Congressional Report* (2015).

[51] David Meyer. 2018. *What the GDPR will mean for companies tracking location.* Retrieved 2022-11-28 from https://iapp.org/news/a/what-the-gdpr-will-mean-for-companies-tracking-location/

[52] Mounira Msahli, Nancy Cam-Winget, William Whyte, Ahmed Serhrouchni, and Houda Labiod. 2020. *RFC 8902 TLS Authentication Using Intelligent Transport System (ITS) Certificates.* Retrieved 2022-11-28 from https://www.hjp.at/doc/rfc/rfc8902.pdf

[53] Toru Nakanishi and Nobuo Funabiki. 2005. Verifier-Local Revocation Group Signature Schemes with Backward Unlinkability from Bilinear Maps. In *Proceedings of the International conference on the theory and application of cryptology and information security*. Chennai, India.

[54] National Highway Traffic Safety Administration. 2017. *Department of Transportation (DOT), Federal Motor Vehicle Safety Standards; V2V Communications.*

Retrieved 2022-11-28 from https://www.federalregister.gov/documents/2017/01/12/2016-31059/federal-motor-vehicle-safety-standards-v2v-communications
[55] Kazuma Ohara, Keita Emura, Goichiro Hanaoka, Ai Ishida, Kazuo Ohta, and Yusuke Sakai. 2019. Shortening the Libert–Peters–Yung Revocable Group Signature Scheme by Using the Random Oracle Methodology. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* (2019), 1101–1117.
[56] Bei Pan, Yu Zheng, David Wilkie, and Cyrus Shahabi. 2013. Crowd Sensing of Traffic Anomalies based on Human Mobility and Social Media. In *Proceedings of the ACM international conference on advances in geographic information systems*. Orlando, Florida.
[57] Orson Peters and Viktor Szakats. 2013. *Portable C implementation of Ed25519*. Retrieved 2022-11-28 from https://github.com/orlp/ed25519
[58] Raluca Popa, Andrew Blumberg, Hari Balakrishnan, and Frank Li. 2011. Privacy and Accountability for Location-Based Aggregate Statistics. In *Proceedings of the ACM conference on computer and communications security*. Chicago, IL, USA.
[59] Sazzadur Rahaman, Long Cheng, Danfeng Yao, He Li, and Jung-Min Park. 2017. Provably Secure Anonymous-yet-Accountable Crowdsensing with Scalable Sublinear Revocation. In *Proceedings of the Privacy Enhancing Technologies*. Minneapolis, USA.
[60] Mina Remeli, Szilvia Lestyán, Gergely Acs, and Gergely Biczók. 2019. Automatic Driver Identification from In-Vehicle Network Logs. In *Proceedings of the IEEE Intelligent Transportation Systems Conference*. Auckland, New Zealand.
[61] Giovanni Rigazzi, Andrea Tassi, Robert Piechocki, Theo Tryfonas, and Andrew Nix. 2017. Optimized Certificate Revocation List Distribution for Secure V2X Communications. In *Proceeding of the IEEE Vehicular Technology Conference*. Sydney, Australia.
[62] Florian Schaub, Frank Kargl, Zhendong Ma, and Michael Weber. 2010. V-Tokens for Conditional Pseudonymity in VANETs. In *Proceedings of the IEEE Wireless Communication and Networking Conference*. Sydney, Australia.
[63] Daniel Slamanig, Raphael Spreitzer, and Thomas Unterluggauer. 2016. Linking-Based Revocation for Group Signatures: A Pragmatic Approach for Efficient Revocation Checks. In *Proceedings of the International Conference on Cryptology in Malaysia*. Kuala Lumpur, Malaysia.
[64] Victor Sucasas, Georgios Mantas, Joaquim Bastos, Francisco Damião, and Jonathan Rodriguez. 2020. A Signature Scheme with Unlinkable-yet-Acountable Pseudonymity for Privacy-Preserving Crowdsensing. *IEEE Transactions on Mobile Computing* (2020), 752–768.
[65] Yipin Sun, Rongxing Lu, Xiaodong Lin, Xuemin Shen, and Jinshu Su. 2010. An Efficient Pseudonymous Authentication Scheme With Strong Privacy Preservation for Vehicular Communications. *IEEE Transactions on Vehicular Technology* (2010), 3589–3603.
[66] Tech Monitor. 2021. *Data from your connected car could be sold to the highest bidder*. Retrieved 2022-11-28 from https://techmonitor.ai/policy/privacy-and-data-protection/connected-vehicle-data-apply-carplay
[67] Patrick Tsang, Man Ho Au, Apu Kapadia, and Sean Smith. 2007. Blacklistable Anonymous Credentials: Blocking Misbehaving Users without TTPs. In *Proceedings of the ACM conference on Computer and communications security*. Alexandria, Virginia, USA.
[68] Patrick P Tsang, Apu Kapadia, Cory Cornelius, and Sean W Smith. 2009. Nymble: Blocking Misbehaving Users in Anonymizing Networks. *IEEE Transactions on Dependable and Secure Computing* 8, 2 (2009).
[69] United States Government Accountability Office. 2017. *Vehicle Data Privacy*. Retrieved 2022-11-28 from https://www.gao.gov/assets/gao-17-656.pdf
[70] Luis Vaquero and Luis Rodero-Merino. 2014. Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing. *ACM SIGCOMM Computer Communication Review* (2014), 27–32.
[71] Eric Verheul, Christopher Hicks, and Flavio Garcia. 2019. IFAL: Issue First Activate Later Certificates for V2X. In *Proceedings of the IEEE European Symposium on Security and Privacy*. Stockholm, Sweden.
[72] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. 2016. AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves. In *Proceedings of the European Symposium on Research in Computer Security*. Heraklion, Greece.
[73] William Whyte, André Weimerskirch, Virendra Kumar, and Thorsten Hehn. 2013. A Security Credential Management System for V2V Communications. In *Proceedings of the IEEE Vehicular Networking Conference*. Boston, MA, USA.
[74] Jiale Zhang, Bing Chen, Yanchao Zhao, Xiang Cheng, and Feng Hu. 2018. Data Security and Privacy-Preserving in Edge Computing Paradigm: Survey and Open Issues. *IEEE Access* (2018), 18209–18237.

A ERCSET CREATION ALGORITHM

We now present in Algorithm 1 the pseudo-code for the algorithm used to implement the createERCSet function, previously described using natural language in Section 4.1. The algorithm receives multiple capabilities to be revoked (from a given pseudonym). It first merges all the latchkeys from the set of provided capabilities (function mergeLatchkeys). Then, the function removeUnsafe will search and remove any latchkeys that may cover time slots that have not been revoked, to ensure that any authentication information that may be used outside the revocation interval, represented by the provided capabilities, is not present in the ERCSet. Afterward, the function removeRedundant will remove any redundant latchkeys, i.e. any latchkey children, that are already covered by their parent. This step is only to achieve efficient storage since it takes advance of our latchkey tree and only requires a logarithmic number of latchkeys to cover the revoked time slots. Finally, the function latchkeysEncoding takes the remaining latchkeys and inserts each one in a Bloom filter that implements the ERCSet. Our prototype implements an optimized version of this algorithm.

Algorithm 1: ERCSet creation, trusted PM side

$C = \langle C^a, \ldots, C^b \rangle$: capabilities to be revoked for a pseudonym

Function mergeLatchkeys(C):
  latchkeySet $\leftarrow \emptyset$ // *latchkeySet* is a genuine set (not a multiset)
  foreach $c_i \in C$ **do**
    latchkeySet $\leftarrow$ *latchkeySet* $\cup$ extractLatchkeys(c_i)
  end
  return *latchkeySet*

Function removeUnsafe($\mathcal{L}$):
  Safe $= \mathcal{L}$
  `/* Remove latchkeys from non-revoked time slots */`
  foreach $l^i \in \mathcal{L}$ **do**
    `/* Returns all children nodes/latchkeys of` l_i `*/`
    $descendants_i \leftarrow$ getLatchkeysSubTree(l_i)
    if $\exists x \in descendants_i \wedge x \notin \mathcal{L}$ **then**
      Safe $\leftarrow$ *Safe* $\setminus l_i$
    end
  end
  return *Safe*

Function removeRedundant($\mathcal{L}$):
  NonRedundantSet $= \mathcal{L}$
  `/* Remove latchkeys that are covered by parent */`
  foreach $l_i \in \mathcal{L}$ **do**
    $parent_i \leftarrow$ getParentLatchkey(l_i)
    if $parent_i \in \mathcal{L}$ **then**
      NonRedundantSet $\leftarrow$ *NonRedundantSet* $\setminus l_i$
    end
  end
  return *NonRedundantSet*

Function latchkeysEncoding($\mathcal{L}$):
  ERCSet $\leftarrow$ createNewBf()
  foreach $l_i \in \mathcal{L}$ **do**
    ERCSet.bfAdd(l_i)
  end
  return *ERCSet*

Function createERCSet(C):
  Enclave Zone Start
  Unfiltered $\leftarrow$ mergeLatchkeys(C)
  Safe $\leftarrow$ removeUnsafe(*Unfiltered*)
  SafeNonRedundant $\leftarrow$ removeRedundant(*Safe*)
  ERCSet $\leftarrow$ latchkeysEncoding(*SafeNonRedundant*)
  Enclave Zone End
  return *ERCSet*